**Ant**

Typically, Ant's build file, called build.xml should reside in the base directory of the project. However there is no restriction on the file name or its location. You are free to use other file names or save the build file in some other location.

<?xml version="1.0"?>

<project name="Hello World Project" default="info">

<target name="info">

<echo>Hello World - Welcome to Apache Ant!</echo>

</target>

</project>

All build files require the project element and at least one target element.

The XML element project has three attributes :

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| name | The Name of the project. (Optional) |
| default | The default target for the build script. A project may contain any number of targets. This attribute specifies which target should be considered as the default. (Mandatory) |
| basedir | The base directory (or) the root folder for the project. (Optional) |

Targets can have dependencies on other targets. For example, a deploy target may have a dependency on the package target, the package target may have a dependency on the compile target and so forth. Dependencies are denoted using the depends attribute. For example:

<target name="deploy" depends="package">

....

</target>

<target name="package" depends="clean,compile">

....

</target>

<target name="clean" >

....

</target>

<target name="compile" >

....

</target>

To run the ant build file, open up command prompt and navigate to the folder where the build.xml resides, and type ant info. You could also type ant instead. Both will work, because info is the default target in the build file.

**Ant - Property Task**

Ant build files are written in XML, which does not allow declaring variables as you do in your favorite programming language. Ant uses the property element which allows you to specify properties. This allows the properties to be changed from one build to another or from one environment to another.

By default, Ant provides the following pre-defined properties that can be used in the build file:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| ant.file | The full location of the build file. |
| ant.version | The version of the Apache Ant installation. |
| basedir | The basedir of the build, as specified in the **basedir**attribute of the **project** element. |
| ant.java.version | The version of the JDK that is used by Ant. |
| ant.project.name | The name of the project, as specified in the **name**atrribute of the **project** element. |
| ant.project.default-target | The default target of the current project. |
| ant.project.invoked-targets | Comma separated list of the targets that were invoked in the current project. |
| ant.core.lib | The full location of the Ant jar file. |
| ant.home | The home directory of Ant installation. |
| ant.library.dir | The home directory for Ant library files - typically ANT\_HOME/lib folder. |

In addition to the above, the user can define additional properties using theproperty element. The following example shows how to define a property called sitename:

<?xml version="1.0"?>

<project name="Hello World Project" default="info">

<property name="sitename" value="www.tutorialspoint.com"/>

<target name="info">

<echo>Apache Ant version is ${ant.version} - You are at ${sitename} </echo>

</target>

</project>

Running Ant on the above build file produces the following output:

C:\>ant

Buildfile: C:\build.xml

info: [echo] Apache Ant version is Apache Ant(TM) version 1.8.2

compiled on December 20 2010 - You are at www.tutorialspoint.com

BUILD SUCCESSFUL

Total time: 0 seconds

C:\>

**Ant - Property Files**

However, for a large project, it makes sense to store the properties in a separate property file. There is no hard and fast rule, but typically the property file is named build.properties and is placed along-side the build.xml file. You could create multiple build properties files based on the deployment environments - such as build.properties.dev and build.properties.test.

The contents of the build property file are similar to the normal java property file. They contain one property per line. Each property is represented by a name and a value pair. The name and value pairs are separated by an equals (=) sign. It is highly recommended that the properties are annotated with proper comments. Comments are listed using the hash (#) character.

The following example shows a build.xml file and its associatedbuild.properties file:

build.xml

<?xml version="1.0"?>

<project name="Hello World Project" default="info">

<property file="build.properties"/>

<target name="info">

<echo>Apache Ant version is ${ant.version} - You are at ${sitename} </echo>

</target>

</project>

build.properties

# The Site Name

sitename=www.tutorialspoint.com

buildversion=3.3.2

**Ant - Data Types**

**Fileset**

The fileset data types represents a collection of files. It is used as a filter to include or exclude files that match a particular pattern.

For example, The fileset selects all .java files in the source folder except those contain the word 'Stub'. The case-sensitive filter is applied.

<fileset dir="${src}" casesensitive="yes">

<include name="\*\*/\*.java"/>

<exclude name="\*\*/\*Stub\*"/>

</fileset>

**Pattern set**

A pattern set is a pattern that allows to filter files or folders easily based on certain patterns.

Patterns can be created using the following meta characters:

* ? - Matches one character only.
* \* - Matches zero or many characters.
* \*\* - Matches zero or many directories recursively.

The following example depicts the usage of a pattern set.

<patternset id="java.files.without.stubs">

<include name="src/\*\*/\*.java"/>

<exclude name="src/\*\*/\*Stub\*"/>

</patternset>

The patternset can then be reused with a fileset as follows:

<fileset dir="${src}" casesensitive="yes">

<patternset refid="java.files.without.stubs"/>

</fileset>

**File list**

The filelist data type is similar to the file set except the following differences:

* filelist contains explicitly named lists of files and it does not support wild cards.
* filelist data type can be applied for existing or non-existing files.

Example

<filelist id="config.files" dir="${webapp.src.folder}">

<file name="applicationConfig.xml"/>

<file name="faces-config.xml"/>

<file name="web.xml"/>

<file name="portlet.xml"/>

</filelist>

**Filter set**

* Using a filterset data type along with the copy task, you can replace certain text in all files that matches the pattern with a replacement value.
* A common example is to append the version number to the release notes file, as shown in the following code.

<copy todir="${output.dir}">

<fileset dir="${releasenotes.dir}" includes="\*\*/\*.txt"/>

<filterset>

<filter token="VERSION" value="${current.version}"/>

</filterset>

</copy>

## Path

The path data type is commonly used to represent a class-path. Entries in the path are separated using semicolons or colons. However, these characters are replaced at the run-time by the executing system's path separator character.

The classpath is set to the list of jar files and classes in the project, as shown in the example below.

<path id="build.classpath.jar">

<pathelement path="${env.J2EE\_HOME}/${j2ee.jar}"/>

<fileset dir="lib">

<include name="\*\*/\*.jar"/>

</fileset>

</path>

In this code:

* The attribute env.J2EE\_HOME points to the environment variableJ2EE\_HOME.
* The attribute j2ee.jar points to the name of the J2EE jar file in the J2EE base folder.